APPENDIX D

"EXPRESS MAIL" Mailing Label Number EI267842785US

Date of Deposit October 24, 1997

I hereby certify under 37 CFR 1.10 that this correspondence is being deposited with the United States Postal Service as **"Express Mail Post Office To Addressee"** with sufficient postage on the date indicated above and is addressed to the Assistant Commissioner for Patents, Washington, D.C. 20231.

Tina Grimstead-Campbell

Tina Grimstead-Campbell

APPENDIX D

Card Class File Converter byte code conversion process

```
/*
 * Reprocess code block.
 */
static
void
reprocessMethod(iMethod* imeth)
{
  int pc;
  int npc;
  int align;
  bytecode* code;
  int codelen;
  int i;
  int opad;
  int npad;
  int apc;
  int high;
  int low;

/* codeinfo is a table that keeps track of the valid Java bytecodes and their
 * corresponding translation
 */
  code = imeth->external->code;
  codelen = imeth->external->code_length;

  jumpPos = 0;
  align = 0;

  /* Scan for unsupported opcodes */
  for (pc = 0; pc < codelen; pc = npc) {
    if (codeinfo[code[pc]].valid == 0) {
      error("Unsupported opcode %d", code[pc]);
    }
    npc = nextPC(pc, code);
  }

  /* Scan for jump instructions an insert into jump table */
  for (pc = 0; pc < codelen; pc = npc) {
    npc = nextPC(pc, code);

    if (codeinfo[code[pc]].valid == 3) {
      insertJump(pc+1, pc, (int16)((code[pc+1] << 8)|code[pc+2]));
    }
    else if (codeinfo[code[pc]].valid == 4) {
      apc = pc & -4;
      low = (code[apc+8] << 24) | (code[apc+9] << 16)
            | (code[apc+10] << 8) | code[apc+11];
      high = (code[apc+12] << 24) | (code[apc+13] << 16)
             | (code[apc+14] << 8) | code[apc+15];
      for (i = 0; i < high-low+1; i++) {
        insertJump(apc+(i*4)+18, pc,
                   (int16)((code[apc+(i*4)+18] << 8) | code[apc+(i*4)+19]));
      }
      insertJump(apc+6, pc, (int16)((code[apc+6] << 8) | code[apc+7]));
    }
    else if (codeinfo[code[pc]].valid == 5) {
      apc = pc & -4;
      low = (code[apc+8] << 24) | (code[apc+9] << 16)
            | (code[apc+10] << 8) | code[apc+11];
      for (i = 0; i < low; i++) {
        insertJump(apc+(i*8)+18, pc,
                   (int16)((code[apc+(i*8)+18] << 8) | code[apc+(i*8)+19]));
      }
      insertJump(apc+6, pc, (int16)((code[apc+6] << 8) | code[apc+7]));
    }
  }
```

A

B

```
#ifdef TRANSLATE_BYTECODE
  /* Translate specific opcodes to general ones */
  for (pc = 0; pc < codelen; pc = npc) {
    /* This is a translation code */
    if (codeinfo[code[pc]].valid == 2) {
      switch (code[pc]) {
      case ILOAD_0:
      case ILOAD_1:
      case ILOAD_2:
      case ILOAD_3:
        insertSpace(code, &codelen, pc, 1);
        align += 1;
        code[pc+1] = code[pc] - ILOAD_0;
        code[pc+0] = ILOAD;
        break;

      case ALOAD_0:
      case ALOAD_1:
      case ALOAD_2:
      case ALOAD_3:
        insertSpace(code, &codelen, pc, 1);
        align += 1;
        code[pc+1] = code[pc] - ALOAD_0;
        code[pc+0] = ALOAD;
        break;

      case ISTORE_0:
      case ISTORE_1:
      case ISTORE_2:
      case ISTORE_3:
        insertSpace(code, &codelen, pc, 1);
        align += 1;
        code[pc+1] = code[pc] - ISTORE_0;
        code[pc+0] = ISTORE;
        break;

      case ASTORE_0:
      case ASTORE_1:
      case ASTORE_2:
      case ASTORE_3:
        insertSpace(code, &codelen, pc, 1);
        align += 1;
        code[pc+1] = code[pc] - ASTORE_0;
        code[pc+0] = ASTORE;
        break;

      case ICONST_M1:
        insertSpace(code, &codelen, pc, 2);
        align += 2;
        code[pc+2] = 255;
        code[pc+1] = 255;
        code[pc+0] = SIPUSH;
        break;

      case ICONST_0:
      case ICONST_1:
      case ICONST_2:
      case ICONST_3:
      case ICONST_4:
      case ICONST_5:
        insertSpace(code, &codelen, pc, 2);
        align += 2;
        code[pc+2] = code[pc] - ICONST_0;
        code[pc+1] = 0;
        code[pc+0] = SIPUSH;
        break;

      case LDC1:
        insertSpace(code, &codelen, pc, 1);
        align += 1;
        code[pc+1] = 0;
        code[pc+0] = LDC2;
        break;
```

```
      case BIPUSH:
        insertSpace(code, &codelen, pc, 1);
        align += 1;
        if ((int8)code[pc+2] >= 0) {
          code[pc+1] = 0;
        }
        else {
          code[pc+1] = 255;
        }
        code[pc+0] = SIPUSH;
        break;

      case INT2SHORT:
        removeSpace(code, &codelen, pc, 1);
        align -= 1;
        npc = pc;
        continue;

      }
    }
    else if (codeinfo[code[pc]].valid == 4 || codeinfo[code[pc]].valid == 5) {
      /* Switches are aligned to 4 byte boundaries.  Since we are inserting and
       * removing bytecodes, this may change the alignment of switch instructions.
       * Therefore, we must readjust the padding in switches to compensate.
       */
      opad = (4 - (((pc+1) - align) % 4)) % 4; /* Current switch padding */
      npad = (4 - ((pc+1) % 4)) % 4;       /* New switch padding */
      if (npad > opad) {
        insertSpace(code, &codelen, pc+1, npad - opad);
        align += (npad - opad);
      }
      else if (npad < opad) {
        removeSpace(code, &codelen, pc+1, opad - npad);
        align -= (opad - npad);
      }
    }

    npc = nextPC(pc, code);
  }
#endif

  /* Relink constants */
  for (pc = 0; pc < codelen; pc = npc) {
    npc = nextPC(pc, code);
    i = (uint16)((code[pc+1] << 8) + code[pc+2]);

    switch (code[pc]) {
    case LDC2:
      /* 'i' == general index */
      switch (cItem(i).type) {
      case CONSTANT_Integer:
        i = cItem(i).v.tint;
        code[pc] = SIPUSH;
        break;

      case CONSTANT_String:
        i = buildStringIndex(i);
        break;

      default:
        error("Unsupported loading of constant type");
        break;
      }
      break;

    case NEW:
    case INSTANCEOF:
    case CHECKCAST:
      /* 'i' == class index */
      i = buildClassIndex(i);
      break;

    case GETFIELD:
    case PUTFIELD:
      /* 'i' == field index */
```

C

D

```
      /* i = buildFieldSignatureIndex(i); */
      i = buildStaticFieldSignatureIndex(i);
      break;

    case GETSTATIC:
    case PUTSTATIC:
      /* 'i' == field index */
      i = buildStaticFieldSignatureIndex(i);
      break;

    case INVOKEVIRTUAL:
    case INVOKENONVIRTUAL:
    case INVOKESTATIC:
    case INVOKEINTERFACE:
      /* 'i' == method signature index */
      i = buildSignatureIndex(i);
      break;
    }

    /* Insert application constant reference */
    code[pc+1] = (i >> 8) & 0xFF;
    code[pc+2] = i & 0xFF;
  }

#ifdef MODIFY_BYTECODE
  /* Translate codes */
  for (pc = 0; pc < codelen; pc = npc) {
    npc = nextPC(pc, code);
    code[pc] = codeinfo[code[pc]].translation;
  }
#endif

  /* Relink jumps */
  for (i = 0; i < jumpPos; i++) {
    apc = jumpTable[i].at;
    pc = jumpTable[i].from;
    npc = jumpTable[i].to - pc;

    code[apc+0] = (npc >> 8) & 0xFF;
    code[apc+1] = npc & 0xFF;
  }

  /* Fixup length */
  imeth->external->code_length = codelen;
  imeth->esize = (SIZEOFMETHOD + codelen + 3) & -4;
}
```

D

E

F